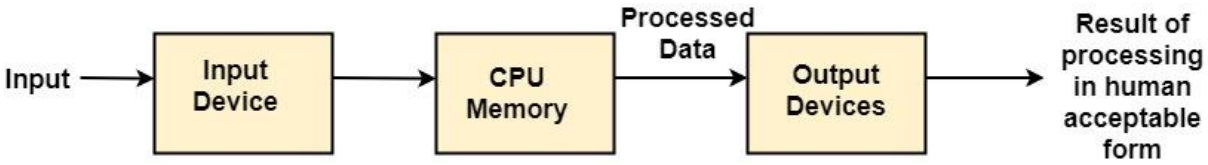
INPUT DEVICES IN COMPUTER GRAPHICS

The Input Devices are the hardware that is used to transfer transfers input to the computer. The data can be in the form of text, graphics, sound, and text. Output device display data from the memory of the computer. Output can be text, numeric data, line, polygon, and other objects.



These Devices include:

1. Keyboard
2. Mouse
3. Trackball
4. Spaceball
5. Joystick
6. Light Pen
7. Digitizer
8. Touch Panels
9. Voice Recognition
10. Image Scanner

SCAN CONVERSION DEFINITION

It is a process of representing graphics objects a collection of pixels. The graphics objects are continuous. The pixels used are discrete. Each pixel can have either on or off state.

The circuitry of the video display device of the computer is capable of converting binary values (0, 1) into a pixel on and pixel off information. 0 is represented by pixel off. 1 is represented using pixel on. Using this ability graphics computer represent picture having discrete dots.

Any model of graphics can be reproduced with a dense matrix of dots or points. Most human beings think graphics objects as points, lines, circles,

ellipses. For generating graphical object, many algorithms have been developed.

Advantage of developing algorithms for scan conversion

1. Algorithms can generate graphics objects at a faster rate.
2. Using algorithms memory can be used efficiently.
3. Algorithms can develop a higher level of graphical objects.

Examples of objects which can be scan converted

1. Point
2. Line
3. Sector
4. Arc
5. Ellipse
6. Rectangle
7. Polygon
8. Characters
9. Filled Regions

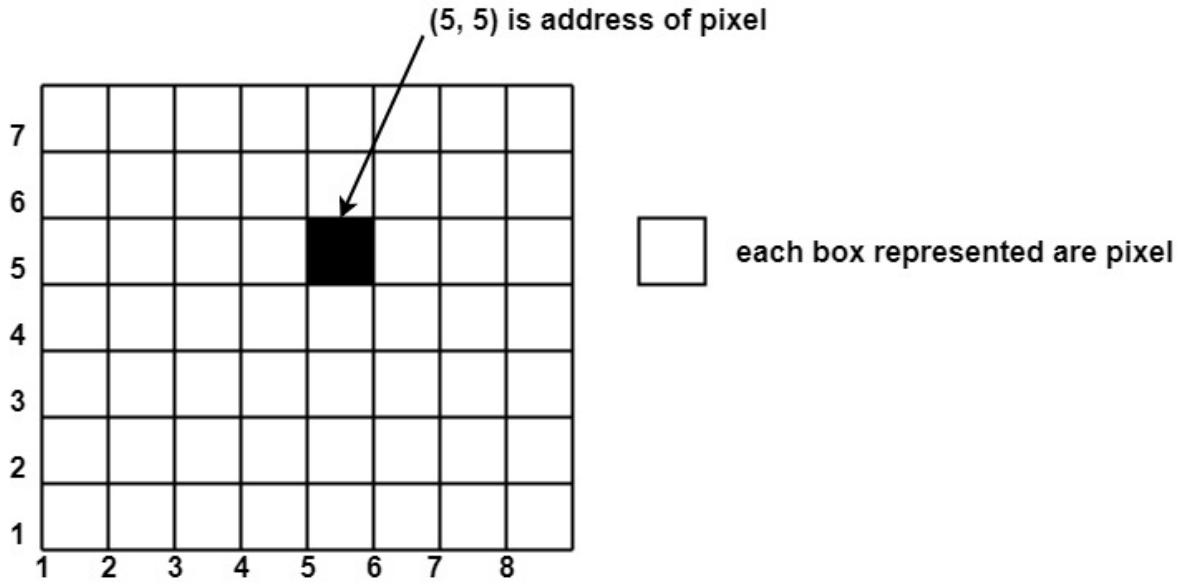
The process of converting is also called as rasterization. The algorithms implementation varies from one computer system to another computer system. Some algorithms are implemented using the software. Some are performed using hardware or firmware. Some are performed using various combinations of hardware, firmware, and software.

Pixel or Pel:

The term pixel is a short form of the picture element. It is also called a point or dot. It is the smallest picture unit accepted by display devices. A picture is constructed from hundreds of such pixels. Pixels are generated using commands. Lines, circle, arcs, characters; curves are drawn with closely spaced pixels. To display the digit or letter matrix of pixels is used.

The closer the dots or pixels are, the better will be the quality of picture. Closer the dots are, crisper will be the picture. Picture will not appear jagged and unclear if pixels are closely spaced. So the quality of the picture is directly proportional to the density of pixels on the screen.

Pixels are also defined as the smallest addressable unit or element of the screen. Each pixel can be assigned an address as shown in fig:



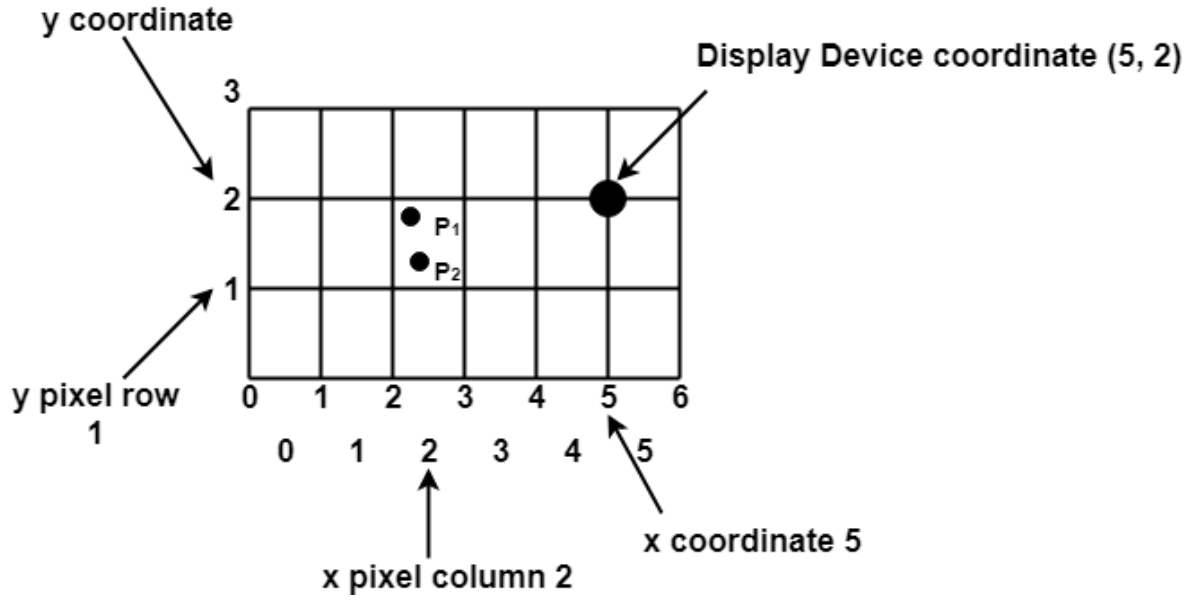
Different graphics objects can be generated by setting the different intensity of pixels and different colors of pixels. Each pixel has some co-ordinate value. The coordinate is represented using row and column.

P (5, 5) used to represent a pixel in the 5th row and the 5th column. Each pixel has some intensity value which is represented in memory of computer called a **frame buffer**. Frame Buffer is also called a refresh buffer. This memory is a storage area for storing pixels values using which pictures are displayed. It is also called as digital memory. Inside the buffer, image is stored as a pattern of binary digits either 0 or 1. So there is an array of 0 or 1 used to represent the picture. In black and white monitors, black pixels are represented using 1's and white pixels are represented using 0's. In case of systems having one bit per pixel frame buffer is called a bitmap. In systems with multiple bits per pixel it is called a pixmap.

Scan Converting a Point

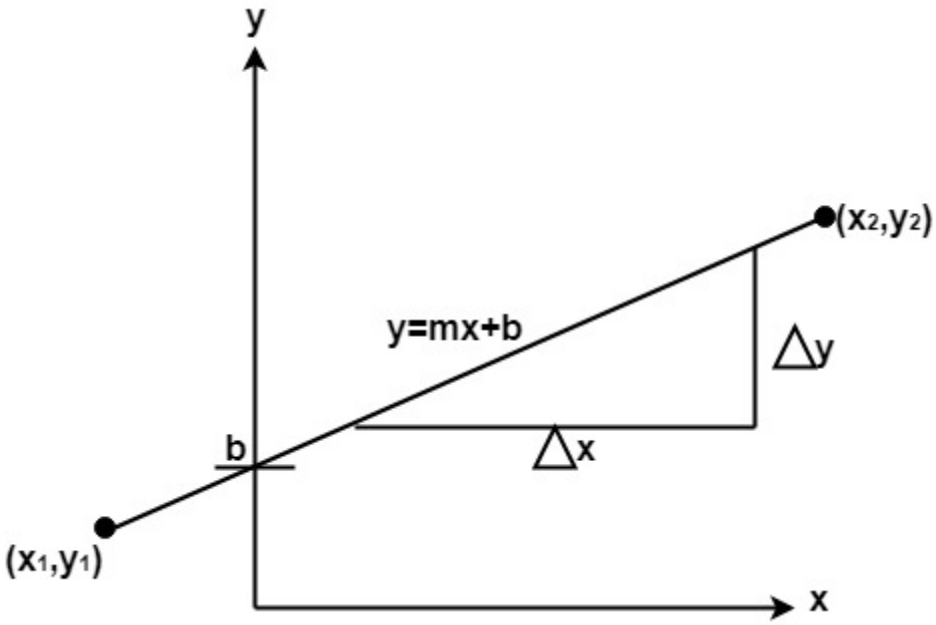
Each pixel on the graphics display does not represent a mathematical point. Instead, it means a region which theoretically can contain an infinite number of points. Scan-Converting a point involves illuminating the pixel that contains the point.

Example: Display coordinates points $P_1\ (2\frac{1}{4}, 1\frac{3}{4})$& $P_2(2\frac{2}{3}, 1\frac{1}{4})$as shown in fig would both be represented by pixel (2, 1). In general, a point p (x, y) is represented by the integer part of x & the integer part of y that is pixels [(INT (x), INT (y).



Scan Converting a Straight Line

A straight line may be defined by two endpoints & an equation. In fig the two endpoints are described by (x_1,y_1) and (x_2,y_2). The equation of the line is used to determine the x, y coordinates of all the points that lie between these two endpoints.



Using the equation of a straight line, $y = mx + b$ where $m = \frac{\Delta y}{\Delta x}$ & b = the y interrupt, we can find values of y by incrementing x from $x = x_1$, to $x = x_2$. By scan-converting these calculated x, y values, we represent the line as a sequence of pixels.

Properties of Good Line Drawing Algorithm:

1. Line should appear Straight: We must appropriate the line by choosing addressable points close to it. If we choose well, the line will appear straight, if not, we shall produce crossed lines.

Fig: O/P from a poor line generating algorithm

The lines must be generated parallel or at 45° to the x and y-axes. Other lines cause a problem: a line segment through it starts and finishes at addressable points, may happen to pass through no another addressable points in between.

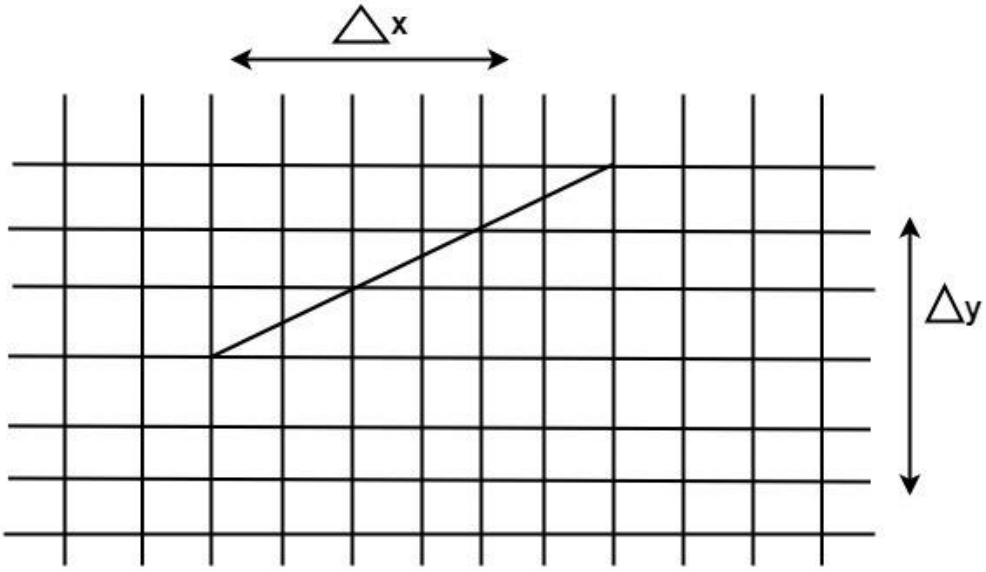


Fig: A straight line segment connecting 2 grid intersection may fail to pass through any other grid intersections.

2. Lines should terminate accurately: Unless lines are plotted accurately, they may terminate at the wrong place.

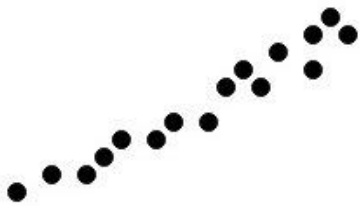


Fig: Uneven line density caused by bunching of dots.

3. Lines should have constant density: Line density is proportional to the no. of dots displayed divided by the length of the line.

To maintain constant density, dots should be equally spaced.

4. Line density should be independent of line length and angle: This can be done by computing an approximating line-length estimate and to use a line-generation algorithm that keeps line density constant to within the accuracy of this estimate.

5. Line should be drawn rapidly: This computation should be performed by special-purpose hardware.

Algorithm for line Drawing:

1. Direct use of line equation
2. DDA (Digital Differential Analyzer)
3. Bresenham's Algorithm

Direct use of line equation:

It is the simplest form of conversion. First of all scan P_1 and P_2 points. P_1 has co-ordinates (x_1',y_1') and (x_2' y_2').

Then m = (y_2',y_1')/(x_2',x_1') and b = $y_1^1 + mx_1^1$

If value of $|m| \leq 1$ for each integer value of x. But do not consider x_1^1 and x_2^2

If value of $|m| > 1$ for each integer value of y. But do not consider y_1^1 and y_2^2

Example: A line with starting point as (0, 0) and ending point (6, 18) is given. Calculate value of intermediate points and slope of line.

Solution: P_1 (0,0) P_7 (6,18)

$x_1=0$
$y_1=0$
$x_2=6$
$y_2=18$

$$M = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{18-0}{6-0} = \frac{18}{6} = 3$$

We know equation of line is

$y = m x + b$
$y = 3x + b$.....equation (1)

put value of x from initial point in equation (1), i.e., (0, 0) x =0, y=0

$0 = 3 \times 0 + b$
$0 = b \Longrightarrow b=0$

put b = 0 in equation (1)

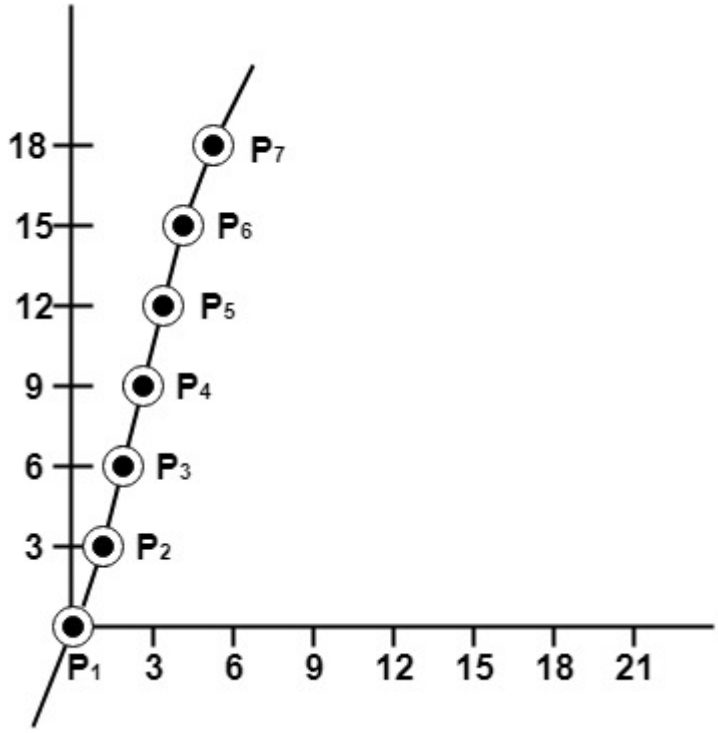
$y = 3x + 0$
$y = 3x$

Now calculate intermediate points

Let $x = 1 \Longrightarrow y = 3 \times 1 \Longrightarrow y = 3$
Let $x = 2 \Longrightarrow y = 3 \times 2 \Longrightarrow y = 6$

Let x = 3 ⟹ y = 3 x 3 ⟹ y = 9
Let x = 4 ⟹ y = 3 x 4 ⟹ y = 12
Let x = 5 ⟹ y = 3 x 5 ⟹ y = 15
Let x = 6 ⟹ y = 3 x 6 ⟹ y = 18

So points are P_1 (0,0)
P_2 (1,3)
P_3 (2,6)
P_4 (3,9)
P_5 (4,12)
P_6 (5,15)
P_7 (6,18)



Algorithm for drawing line using equation:

Step1: Start Algorithm

Step2: Declare variables x_1,x_2,y_1,y_2,dx,dy,m,b,

Step3: Enter values of x_1,x_2,y_1,y_2.
The (x_1,y_1) are co-ordinates of a starting point of the line.
The (x_2,y_2) are co-ordinates of a ending point of the line.

Step4: Calculate dx = x_2- x_1

Step5: Calculate dy = y_2-y_1

Step6: Calculate m = $\frac{dy}{dx}$

Step7: Calculate b = y_1-m* x_1

Step8: Set (x, y) equal to starting point, i.e., lowest point and x_{end}equal to largest value of x.

If dx < 0

then $x = x_2$

$y = y_2$

$x_{end} = x_1$

If dx > 0

then $x = x_1$

$y = y_1$

$x_{end} = x_2$

Step9: Check whether the complete line has been drawn if $x=x_{end}$, stop

Step10: Plot a point at current (x, y) coordinates

Step11: Increment value of x, i.e., x = x+1

Step12: Compute next value of y from equation y = mx + b

Step13: Go to Step9.

Program to draw a line using LineSlope Method

```
#include <graphics.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>

class bresen
{
    float x, y, x1, y1, x2, y2, dx, dy, m, c, xend;
    public:
    void get ();
    void cal ();
};
    void main ()
    {
    bresen b;
    b.get ();
    b.cal ();
    getch ();
```

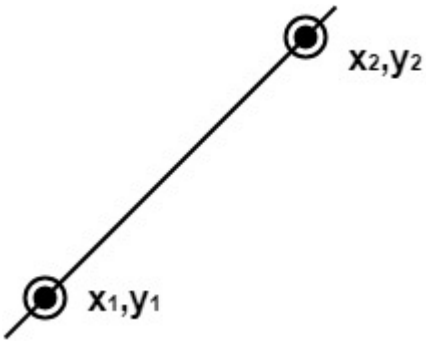
```
21.  }
22.   Void bresen :: get ()
23.  {
24.   print ("Enter start & end points");
25.   print ("enter x1, y1, x2, y2");
26.   scanf ("%f%f%f%f",sx1, sx2, sx3, sx4)
27.}
28.void bresen ::cal ()
29.{
30.   /* request auto detection */
31.   int gdriver = DETECT,gmode, errorcode;
32.   /* initialize graphics and local variables */
33.   initgraph (&gdriver, &gmode, " ");
34.   /* read result of initialization */
35.   errorcode = graphresult ();
36.   if (errorcode ! = grOK)    /*an error occurred */
37.   {
38.      printf("Graphics error: %s \n", grapherrormsg (errorcode);
39.      printf ("Press any key to halt:");
40.      getch ();
41.      exit (1); /* terminate with an error code */
42.   }
43.   dx = x2-x1;
44.   dy=y2-2y1;
45.   m = dy/dx;
46.   c = y1 - (m * x1);
47.   if (dx<0)
48.   {
49.      x=x2;
50.      y=y2;
51.      xend=x1;
```

```
52.   }
53.   else
54.   {
55.      x=x1;
56.      y=y1;
57.      xend=x2;
58.   }
59.while (x<=xend)
60.   {
61.      putpixel (x, y, RED);
62.      y++;
63.      y=(x*x) +c;
64.   }
65.}
```

OUTPUT:

Enter Starting and End Points

Enter (X1, Y1, X2, Y2) 200 100 300 200



DDA ALGORITHM

DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps.

Suppose at step i, the pixels is (x_i,y_i)

The line of equation for step i

$y_i=mx_{i+b}$.....equation 1

Next value will be

$y_{i+1}=mx_{i+1}+b$.....equation 2

$m=\frac{\Delta y}{\Delta x}$

$y_{i+1}-y_i=\Delta y$.....equation 3

$y_{i+1}-x_i=\Delta x$.....equation 4

$y_{i+1}=y_i+\Delta y$

$\Delta y=m\Delta x$

$y_{i+1}=y_i+m\Delta x$

$\Delta x=\Delta y/m$

$x_{i+1}=x_i+\Delta x$

$x_{i+1}=x_i+\Delta y/m$

Case1: When $|M|<1$ then (assume that $x_1<x_2$)

$x=x1, y=y_1$ set $\Delta x=1$

$y_{i+1}=y_{1+m}$, $x=x+1$

Until $x = x_2</x$

Case2: When $|M|<1$ then (assume that $y1<y_2$)

$x=x1, y=y_1$ set $\Delta y=1$

$x_{i+1}=\frac{1}{m}$, $y=y+1$

Until $y \rightarrow y_2</y$

Advantage:

1. **It is a faster method than method of using direct use of line equation.**
2. **This method does not use multiplication theorem.**
3. **It allows us to detect the change in the value of x and y ,so plotting of same point twice is not possible.**
4. **This method gives overflow indication when a point is repositioned.**
5. **It is an easy method because each step involves just two additions.**

Disadvantage:

1. **It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.**
2. **Rounding off operations and floating point operations consumes a lot of time.**
3. **It is more suitable for generating line using the software. But it is less suited for hardware implementation.**

DDA Algorithm:

Step1: Start Algorithm

Step2: Declare x_1,y_1,x_2,y_2,dx,dy,x,y as integer variables.

Step3: Enter value of x_1,y_1,x_2,y_2.

Step4: Calculate dx = x_2-x_1

Step5: Calculate dy = y_2-y_1

Step6: If ABS (dx) > ABS (dy)
Then step = abs (dx)
Else

Step7: x_{inc}=dx/step
y_{inc}=dy/step
assign x = x_1
assign y = y_1

Step8: Set pixel (x, y)

Step9: x = x + x_{inc}
y = y + y_{inc}
Set pixels (Round (x), Round (y))

Step10: Repeat step 9 until x = x_2

Step11: End Algorithm

Example: If a line is drawn from (2, 3) to (6, 15) with use of DDA. How many points will needed to generate such line?

Solution: P_1 (2,3) P_{11} (6,15)

x_1=2
y_1=3
x_2= 6
y_2=15
dx = 6 - 2 = 4
dy = 15 - 3 = 12
m = $\frac{dy}{dx} = \frac{12}{4}$

For calculating next value of x takes x = x + $\frac{1}{m}$

$P_1(2, 3)$ point plotted

$P_2(2\frac{1}{3}, 4)$ point plotted

$P_3(2\frac{2}{3}, 5)$ point not plotted

$P_4(3, 6)$ point plotted

$P_5(3\frac{1}{3}, 7)$ point not plotted

$P_6(3\frac{2}{3}, 8)$ point not plotted

$P_7(4, 9)$ point plotted

$P_8(4\frac{1}{3}, 10)$ point not plotted

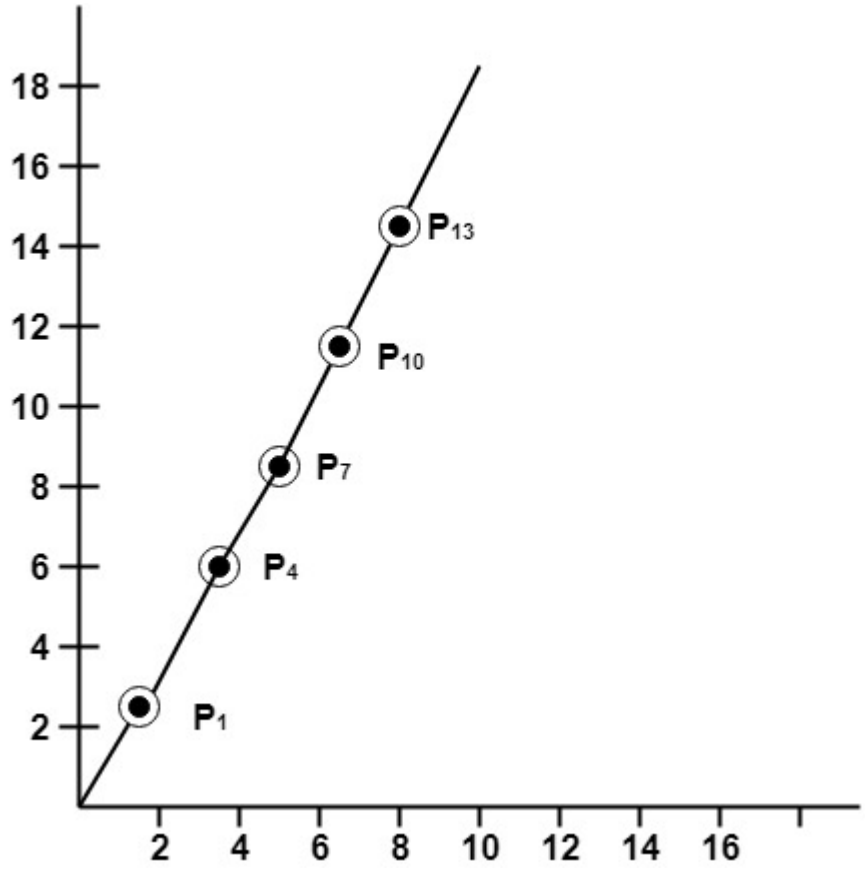
$P_9(4\frac{2}{3}, 11)$ point not plotted

$P_{10}(5, 12)$ point plotted

$P_{11}(5\frac{1}{3}, 13)$ point not plotted

$P_{12}(5\frac{2}{3}, 14)$ point not plotted

$P_{13}(6, 15)$ point plotted



Program to implement DDA Line Drawing Algorithm:

1. #include<graphics.h>
2. #include<conio.h>
3. #include<stdio.h>
4. **void** main()

```
5. {
6.     intgd = DETECT ,gm, i;
7.     float x, y,dx,dy,steps;
8.     int x0, x1, y0, y1;
9.     initgraph(&gd, &gm, "C:\\TC\\BGI");
10.          setbkcolor(WHITE);
11.          x0 = 100 , y0 = 200, x1 = 500, y1 = 300;
12.          dx = (float)(x1 - x0);
13.          dy = (float)(y1 - y0);
14.          if(dx>=dy)
15.               {
16.             steps = dx;
17.          }
18.          else
19.               {
20.             steps = dy;
21.          }
22.          dx = dx/steps;
23.          dy = dy/steps;
24.          x = x0;
25.          y = y0;
26.          i = 1;
27.          while(i<= steps)
28.          {
29.             putpixel(x, y, RED);
30.             x += dx;
31.             y += dy;
32.             i=i+1;
33.          }
```

```
34.         getch();
35.         closegraph();
36.     }
```

Output:

Symmetrical DDA:

The Digital Differential Analyzer (DDA) generates lines from their differential equations. The equation of a straight line is

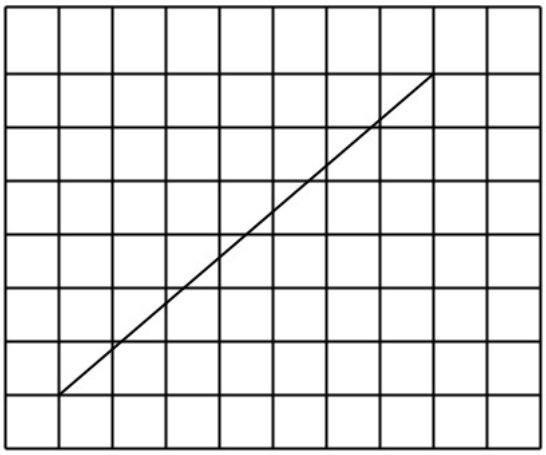
$$\frac{dy}{dx} = \frac{\Delta y}{\Delta x}$$

The DDA works on the principle that we simultaneously increment x and y by small steps proportional to the first derivatives of x and y. In this case of a straight line, the first derivatives are constant and are proportional to Δx and Δy . Therefore, we could generate a line by incrementing x and y by $\epsilon \Delta x$ and $\epsilon \Delta y$, where ϵ is some small quantity. There are two ways to generate points

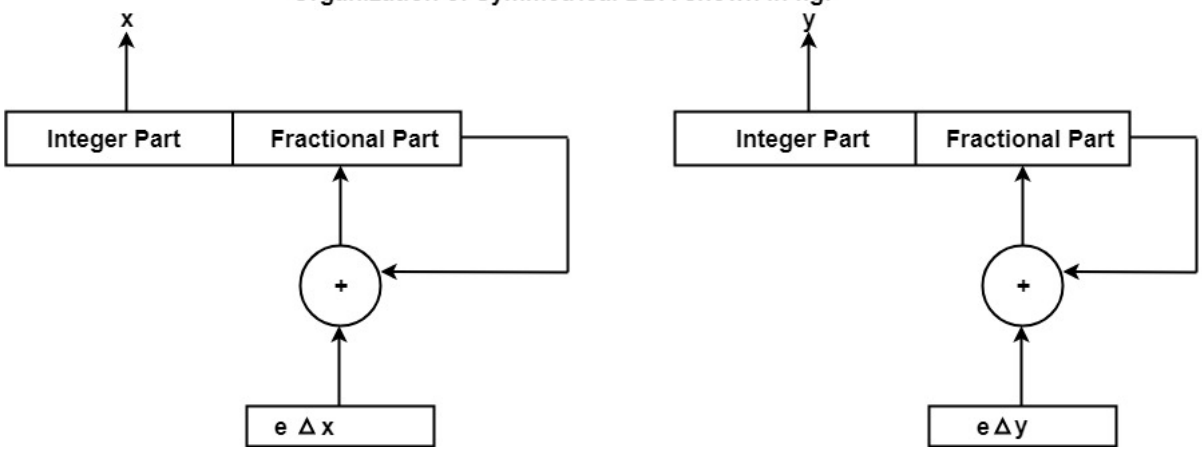
1. By rounding to the nearest integer after each incremental step, after rounding we display dots at the resultant x and y.

2. An alternative to rounding the use of arithmetic overflow: x and y are kept in registers that have two parts, integer and fractional. The incrementing values, which are both less than unity, are repeatedly added to the fractional parts and whenever the results overflows, the corresponding integer part is incremented. The integer parts of the x and y registers are used in plotting the line. In the case of the symmetrical DDA, we choose $\varepsilon=2^{-n}$,where $2^{n-1} \leq \max(|\Delta x|,|\Delta y|) < 2^{\pi}$

A line drawn with the symmetrical DDA is shown in fig:



Organization of Symmetrical DDA shown in fig:



Example: If a line is drawn from (0, 0) to (10, 5) with a symmetrical DDA

1. How many iterations are performed?

2. How many different points will be generated?

Solutions: Given: P^1 (0,0) P^2 (10,5)

$x^1=0$

$y^1=0$

$x^2=10$

$y^2=5$

dx = 10 - 0 = 10

dy = 5 - 0 = 0

$$m = \frac{y_2-y_1}{x_2-x_1} = \frac{5-0}{10-0} = \frac{5}{10} = 0.5$$

P^1 (0,0) will be considered starting points

P^3 (1,0.5) point not plotted

P^4 (2, 1) point plotted

P^5 (3, 1.5) point not plotted

P^6 (4,2) point plotted

P^7 (5,2.5) point not plotted
P^8 (6,3) point plotted
P^9 (7,3.5) point not plotted
P^{10} (8, 4) point plotted
P^{11} (9,4.5) point not plotted
P^{12} (10,5) point plotted

Following Figure show line plotted using these points.